Project 1

Write an AVR assembly program to code RC5.

Introduction

RC5 is designed by Rivest, which is a symmetric-key block cipher notable for its simplicity (http://people.csail.mit.edu/rivest/pubs/Riv94.pdf). A key feature of RC5 is the heavy use of data-dependent rotations. RC5 has a variable word size, a variable number of rounds, and a variable length secret key. This section presents the encryption, decryption, and key-expansion based on RC5 algorithm with the following parameters:

- the number of rounds (r) equals 8,
- the size of the expanded key table ($t = 2*(r+1)$) equals 18,
- the word size in bits (w) equals 16,
- the word size in bytes ($u = w/8$) equals 2,
- the number of bytes in the secrete key (b) equals 12,
- the number of words in the secrete key ($c = \lceil b/u \rceil$) equals 6,
- the number of iterations of the key-expansion module ($n = 3*\max(t, c)$) equals 54,
- the constant P_{16} used in the key-expansion module equals $(b7e1)_{16}$ or $(1011011111100001)_2$, where $P_w = \mathrm{Odd}((e-2)*2^w$ and $e = 2.718281828459$, and
- the constant Q_{16} used in the key-expansion module equals $(9e37)_{16}$ or $(1001111000110111)_2$, where $Q_w = \mathrm{Odd}((\phi-1)*2^w$ and $\phi = 1.618033988749$.

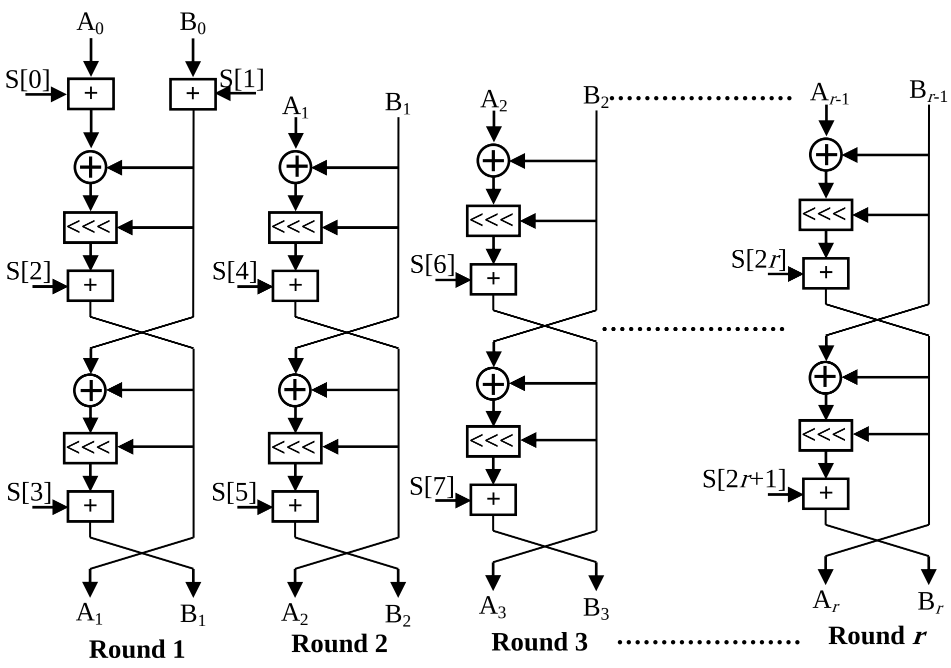

Fig. 1. Unrolled RC5 encryption algorithm with r rounds.

RC5 Encryption

The encryption module of RC5 accepts a block of data in two w-bit inputs A_0 and B_0, as shown in Figure 1. Moreover, it accepts the expanded key array $S[0{:}t\text{-}1]$, which stores the round keys generated by the key-expansion module. After r rounds, the encryption module generates an encrypted block in two w-bit outputs A_r and B_r. Listing 1 presents the pseudo-code of the RC5 encryption algorithm, where the main operations are addition (+), XOR (⊕) and shift-left (<<<).

Listing 1: RC5 encryption algorithm

$A_0 = A_0 + S[0]$
$B_0 = B_0 + S[1]$
for i = 1 to r
$\quad A_i = ((A_{i\text{-}1} \oplus B_{i\text{-}1}) <<< B_{i\text{-}1}) + S[2*i]$
$\quad B_i = ((B_{i\text{-}1} \oplus A_i) <<< A_i) + S[2*i+1]$

Figure 1 shows the block diagram of the unrolled encryption module of RC5 algorithm, where a single clock cycle is required for encrypting 2×w-bit. It is clear that (2r+2)×w-bit adders, 2r×w-bit shift-left, and 2r×w-bit XOR are needed for implementing the encryption module with unrolling r rounds.

RC5 Decryption

By reversing the operations, the decryption process can be easily derived from the encryption algorithm. Listing 2 presents the pseudo-code of the RC5 decryption algorithm, where the main operations are subtraction (–), XOR (⊕) and shift-right (>>>).

Listing 2: RC5 decryption algorithm

for i = r downto 1
$\quad B_{i\text{-}1} = ((B_i - S[2*i+1] >>> A_i) \oplus A_i$
$\quad A_{i\text{-}1} = ((A_i - S[2*i] >>> B_{i\text{-}1}) \oplus B_{i\text{-}1}$
$B_0 = B_0 - S[1]$
$A_0 = A_0 - S[0]$

Like encryption, the unrolled decryption of RC5 algorithm is implemented to decrypt 2×w-bit in a single clock cycle. (2r+2)×w-bit subtractors, 2r×w-bit shift-right, and 2r×w-bit XOR are needed for implementing the decryption module with unrolling r rounds.

RC5 Key-Expansion

The key-expansion module expands the user's secret key K to fill the expanded key array S, where S resembles an array of t = 2*(r +1) random binary words determined by K. The key-expansion algorithm uses two "magic constants": P_w = Odd((e – 2)*2^w and Q_w = Odd((ϕ – 1)*2^w, where e is the base of natural logarithms (2.718281828459), ϕ is the golden ratio

(1.618033988749), and Odd(x) is the odd integer nearest to x. For w = 16, P_{16} equals $(b7e1)_{16}$ or $(1011011111100001)_2$, and Q_{16} equals $(9e37)_{16}$ or $(1001111000110111)_2$.

As discussed in (http://people.csail.mit.edu/rivest/pubs/Riv94.pdf), the key-expansion algorithm consists of three simple algorithmic parts, see Listing 3. The first step is to copy the secret key $K[0: b\text{-}1]$ into an array $L[0: c\text{-}1]$, where b is the number of bytes in the secrete key, c is the number of words in the secrete key ($c = \lceil b/u \rceil$), and u is the number of bytes per word. Note that any unfilled byte positions of L are zeroed. The second step is to initialize array S to a particular fixed (key-independent) pseudo-random bit pattern, using an arithmetic progression modulo 2^w determined by the "magic constants" P_w and Q_w, where $S[0] = P_w$ and $S[i] = S[i-1] + Q_w$, for i = 1 to t – 1. Finally, the third step of key-expansion is to mix in the user's secret key in three passes over the arrays S and L. More precisely, due to the potentially different sizes of S and L, the larger array will be processed three times, and the other may be handled more times.

Listing 3: Key-expansion algorithm

```
// First step
    for i = b – 1 downto 0
        L[i / u] = (L[i / u] <<< 8) + K[i];
// Second step
    S[0] = P_w
    for i = 1 to t – 1
        S[i] = S[i – 1] + Q_w
// Third step
    i = j = 0
    A = B = 0
    do 3*max(t, c) times
        A = S[i] = (S[i] + A + B) <<< 3
        B = L[j] = (L[j] + A + B) <<< (A + B)
        i = (i + 1) mod(t)
        j = (j + 1) mod(c)
```